VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Nástroj pro zálohování, transformaci a obnovu dat v relační databázi

Tool for Backup, Transformation and Restoration of Data in Relational Databases

2013 Martin Buček

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Martin Buček**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Nástroj pro zálohování, transformaci a obnovu dat v relační databázi
Tool for Backup, Transformation and Restoration of Data in Relational Databases

Zásady pro vypracování:

Cílem je vytvořit nástroj, který umožní zálohování, transformaci a obnovu dat v relační databázi použitím některého z formátů snadno upravitelných přímo člověkem pomocí textového editoru. Tento nástroj má především pomoci při přenesení dat po přegenerování schématu pomocí ORM nástroje, manipulaci se schématem databáze tedy řešte do úrovně vhodné k dosažení tohoto cíle.
Prostředí a použitý databázový stroj pro implementaci zvolte po konzultaci s vedoucím, implementaci provádějte s ohledem na rozšířitelnost pro další databázové stroje.

1. Zhodnoťte běžné strukturované textové formáty z hlediska vhodnosti pro tuto aplikaci. Vyberte jeden z formátů pro další implementaci.
2. Vytvořte nástroj, který bude schopen z existující databáze zálohovat data do zvoleného formátu a také je obnovit.
3. Rozšiřte tento nástroj o vhodné možnosti transformace dat. Umožněte takto ošetřit situaci, kdy proti původnímu schématu sloupec ubyl či přibyl.
4. Připravte a popište ukázkový případ použití.

Seznam doporučené odborné literatury:

[1] Esposito, Dino. XML : efektivní programování pro .NET. Grada Publishing, 2004. ISBN 80-247-0775-6
[2] Rys, Michael. "XML and relational database management systems: inside Microsoft® SQL Server™ 2005." Proceedings of the 2005 ACM SIGMOD international conference on Management of data. ACM, 2005
[3] Ben-Kiki, Oren, Clark Evans, and Brian Ingerson. "YAML Ain't Markup Language (YAML™) Version 1.1." Working Draft 2008-05 11 (2001)
[4] Transact-SQL Reference <http://msdn.microsoft.com/en-us/library/bb510741.aspx>

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jakub Macek**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2013 ..... Martin Bučík

Rád bych poděkoval panu Ing. Jakubu Mackovi za odbornou pomoc a konzultaci při vytváření této bakalářské práce.

Abstrakt

Cílem bakalářské práce je vytvořit nástroj, který umožní uživateli zálohovat data z relační databáze do zvoleného textového formátu a obnovit data zpět do databáze. Nástroj je rozšířen o možnost transformace k ošetření situací, pokud sloupec ubyl nebo přibyl. Zároveň je nástroj naimplementován s ohledem na rozšiřitelnost o další databázové stroje. V tomto textu jsou diskutovány vhodné textové formáty pro další implementaci, transformace relační databáze a názorný příklad použití.

Klíčová slova: bakalářská práce, zálohování, obnova, transformace, databázové systémy

Abstract

The objective of thesis is to create tool, which allows the user to backup data from relational database to the chosen text format and restore data back to this database. The tool is extended to transformation for situation when column was added or deleted. Simultaneously this tool is implemented with regard for extension of other database engines. In this text are discussed suitable text formats for further implementation, transformation of relational database and an sample example for usage.

Keywords: thesis, backup, restoration, transformation, database systems

Seznam použitých zkratek a symbolů

AJAX	–	Asynchronous JavaScript and XML
ANSI	–	American National Standards Institute
CLI	–	Command Line Interface
DLL	–	Dynamic Link Library
DTD	–	Document Type Definition
HTML	–	Hyper Text Markup Language
IBM	–	International Business Machines
INGRES	–	Interactive Graphics and Retrieval System
ISO	–	International Organization for Standardization
JSON	–	JavaScript Object Notation
PDF	–	Portable Document Format
QMF	–	Query Management Facility
SGML	–	Standard Generalized Markup Language
SOAP	–	Simple Object Access Protocol
SQL	–	Structured Query Language
SŘBD	–	Systém řízení báze dat
W3C	–	World Wide Web Consortium
XML	–	eXtensible Markup Language
YAML	–	YAML Ain't Markup Language

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Bakalářská práce je rozdělena do tří hlavních kapitol. V první kapitole je vysvětlen výběr textového formátu pro serializaci. Postupně jsou vysvětleny výhody a nevýhody jednotlivých textových formátů a ke každému z nich je pro lepší pochopení uveden příklad daného dokumentu v diskutovaném formátu a jeho syntaxe. Druhá kapitola je zaměřena na praktičtější část. Krok po kroku je zde vysvětlena práce s nástrojem, kterou si může čtenář vyzkoušet sám na ukázkové databázi. V poslední kapitole bude čtenář detailněji obeznámen s použitou transformací v ukázkovém příkladě.

2 Textové serializační formáty

Cílem této kapitoly je zhodnotit výhody a nevýhody textových formátů z hlediska dosažení cílů bakalářské práce a vybrat jeden vhodný pro serializaci. Jako potencionální kandidáti byly vybrány formáty XML, JSON a YAML. Tyto formáty byly vybrány na základě četnosti jejich využití pro serializaci. Minimální požadavek pro výběr formátů byl fakt, aby daný formát byl textový a ne binární. To z důvodu, aby byl soubor upravitelný v libovolném textovém editoru.

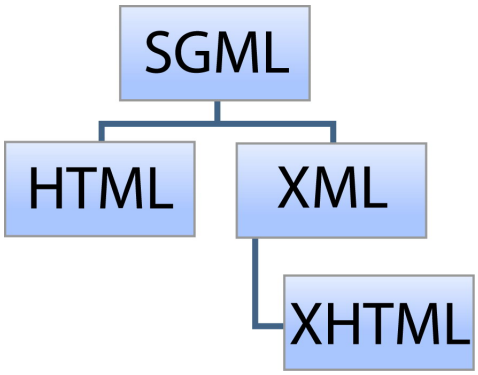
2.1 XML

2.1.1 Původ

XML byl vytvořen skupinou XML working group a standartizován pod záštitem konsorcia W3C 10.ledna 1998 ve verzi 1.0. XML byl vydán celkem ve dvou verzích. Druhá verze 1.1 byla vydána v 4.února 2004. Zásadní rozdíl oproti starší verzi 1.0 tkví v požadavcích na korektní názvy elementů a atributů. Částečně se tímto problémem detailněji zabývá jedna z dalších podkapitol.

XML byl navržen k řešení problémů elektronického publikování a v současnosti hraje velice důležitou roli při výměně nejrůznějších údajů na webu i jinde. V dnešní době je základem velmi populárních webových služeb, SOAP protokolu a také slouží pro ukládání informací databázového typu.

XML vychází ze značkovacího jazyka SGML. Základní rozdíl mezi SGML a XML spočívá ve zpracování daného dokumentu.XML pouze specifikuje, co se nachází v dokumentu. Oproti HTML neříká nic o tom, jak má daný dokument vypadat. Nicméně, tyto značkovací jazyky spolu úzce souvisí.



Obrázek 1: Značkovací jazyky vycházející z SGML

2.1.2 Výhody a nevýhody

Z velké části byla náplní mé bakalářské práce manipulace s daty v textovém formátu. Tento fakt hrál klíčovou roli ve výběru textového formátu pro iplementaci. XML má

mnoho výhod, nicméně já zde budu popisovat pouze ty výhody, které byly důležité pro dosažení cíle bakalářské práce.

XML je velice rozšířené s širokou podporou v mnoha programovacích jazycích a vývojových prostředích. Výhodou je možnost použití třídy XmlTextWriter, který je součástí .NET frameworku už od verze 1.0. XmlTextWriter představuje třídu k rychlému zapisování dat do XML formátu. Během zápisu se data neukládájí do paměti, což dělá zapisování rychlé. Další výhodou, která mě přesvědčila k výběru toho formátu k další iplementaci, je možnost vytvoření si vlastních tagů.

Nutno podotknout, že XML má i stinné stránky. Mezi jeho nevýhody z hlediska této práce patří zejména několikanásobně větší velikost než u ostatních formátů.

2.1.3 Základy syntaxe

Při otevření XML dokumentu se můžeme potkat s různými prvky. XML dokument může začínat prologem. Prolog specifikuje verzi XML a použité kódování. Nicméně přítomnost prologu v dokumentu není nutná.

```
<?xml version="1.0" encoding="UTF-8" ?>
```

Základním stavebním prvkem každého XML dokumentu jsou tagy. Rozeznáváme několik druhů.

- Počáteční tag:

  ```
  <Jmeno>
  ```

- Koncový tag:

  ```
  </Jmeno>
  ```

- Prázdný tag:

  ```
  <Jmeno />
  ```

Mezi počátečním a koncovým tagem se nachází obsah. Dohromady tyto 3 části tvoří element. Ovšem element je také prázdný tag sám o sobě. Elementy se mohou vnořovat i do jiných elementů a tím může vzniknout poměrně rozsáhlý dokument. Ukázku podobné struktury dokumentu můžete vidět v dalších podkapitolách této práce.

Dalším prvkem XML jsou atributy. Atributy slouží pro oddělení dat. Atributy se zapisují do počátečních nebo prázdných tagů. V jednom tagu jich může být i více za sebou. Obvykle se používájí pro uložení informací, které nejsou pro čtenáře až tak důležité. Nutno podotknout, že při implementaci nebyly atributy v této bakalářské práci použity. Příkladem atributu může být třeba nějaké identifikační číslo.

```
<Jmeno id="buc0022">Martin</Jmeno>
```

```
<Jmeno id="buc0022" />
```

Posledním prvkem syntaxe XML dokumentu jsou komentáře. Komentáře se mohou objevit kdekoliv v dokumentu mimo tagy. Také nesmí být před XML prologem.

```
<!-- Toto je nový komentář -->
```

2.1.4 Správně strukturovaný dokument

Znalost jednotlivých konstrukčních prvků úplně nestačí. XML dokument musí splňovat několik pravidel, aby se dal označit na správně strukturovaný nebo-li „well-formed“ dokument. Správně strukturovaný dokument znamená, že splňuje všechny požadavky kladené na syntaxi, které jsou definovány ve specifikaci XML 1.0.

- XML dokument musí obsahovat právě jeden kořenový element, do kterého jsou vnořeny všechny ostatní elementy.
- Každý začínající tag musí mít odpovídající koncový tag.
- Elementy se nesmí křížit. Typický příklad křížení může být třeba tento:

  ```
  <Jmeno><Prijmeni></Jmeno></Prijmeni>
  ```

- Hodnoty všech atributů musí být uzavřeny v úvozovkách.
- Všechny znaky, které se v dokumentu vyskytují musí podléhat řádnému Unicode kódování.

XML nabízí možnost vytvoření vlastních tagů. Nicméně je důležité znát pár pravidel pro vytvoření korektního elementu. Název elementu nesmí obsahovat žádné mezery a také různé speciální znaky. Názvy elementů jsou také case-sensitive. To znamená, že rozeznává malá a velká písmena. Například následující zápis by vyústil v chybu.

```
<Jmeno>Martin</jmeno>
```

Správný zápis těchto elementů bude vypadat následovně.

```
<Jmeno>Martin</Jmeno>
```

Ruční kontrola validace méně rozsáhlého XML dokumentu nemusí být nějak zdlouhává. Ovšem u větších dokumentů už by to mohl být problém. Pro tyto situace existují různé online nástroje. Odkaz na jeden z nich najdete v použité literatuře na konci této bakalářské práce.

login	jméno	příjmení	email
buc0022	Martin	Buček	martinbucek@seznam.cz
nov0011	Jan	Novák	jannovak@gmail.com

Tabulka 1: SQL tabulka uživatel

2.1.5 Příklad dokumentu

Uvažme pro tento příklad tuto jednoduchou tabulku uživatelů, která se nachází v relační databázi na Microsoft SQL Serveru. Pro serializaci použijeme již výše zmíněný XmlTextWriter.

Na výstupu dostaneme tento XML dokument. Můžeme si všimnout lehce upravené struktury dokumentu. Do celé struktury dokumentu jsou přidány ještě úrovně pro oddělení jednotlivých záznamů v tabulce, čehož později využijeme u obnovy dat. Navíc všechny požadované data určené k zálohování jsou uvnitř kořenového elementu s názvem Databaze. To splňuje jeden z požadavků na korektně vytvořený XML dokument.

V předešlé podkapitole je zmíňka o online nástrojích určené k validaci XML dokumentů. Můžete si zkusit zkopírovat tento XML dokument a použít ho jako demonstrační příklad k ověření. Pouze chci upozornit, že při kopírování z PDF to vynechává podtržítka v názvech elementů id uživatele. Proto je nutné je dopsat ručně.

```
<Databaze>
  <Uzivatel>
    <Radek>
      <id_uzivatele>buc0022</id_uzivatele>
      <jmeno>Martin</jmeno>
      <prijmeni>Buček</prijmeni>
      <email>martinbucek@seznam.cz</email>
    </Radek>
    <Radek>
      <id_uzivatele>nov0011</id_uzivatele>
      <jmeno>Jan</jmeno>
      <prijmeni>Novák</prijmeni>
      <email>jannovak@gmail.com</email>
    </Radek>
  </Uzivatel>
</Databaze>
```

Výpis 1: XML dokument

Funkce pro serializaci do XML formátu vypadá následovně.

```
using (XmlTextWriter writer = new XmlTextWriter(stream, Encoding.UTF8))
{
        writer.Formatting = System.Xml.Formatting.Indented;
        writer.WriteStartElement("Databaze");
        command = new SqlCommand();
        foreach (string tableName in list)
        {
                command = new SqlCommand("Select * from "+tableName, connection);
                adapter = new SqlDataAdapter(command);
```

```
10              DataTable tableOfContents = new DataTable();
11              adapter. Fill (tableOfContents);
12              string  tableName2 = tableName.Replace(" ", "--");
13              writer .WriteStartElement(tableName2);
14              foreach (DataRow row in tableOfContents.Rows)
15              {
16                      writer .WriteStartElement("Radek");
17                      foreach (DataColumn column in tableOfContents.Columns)
18                      {
19                              writer .WriteStartElement(column.ColumnName);
20                              writer .WriteString(row[column].ToString().Trim());
21                              writer .WriteEndElement();
22                      }
23                      writer .WriteEndElement();
24              }
25              writer .WriteEndElement();
26      }
27      writer .WriteEndElement();
28      writer .Close();
29  }
```

Výpis 2: Serializace do XML formátu

K zápisu použijeme výše zmíněný XmlTextWriter, který jako parametr přijímá FileStream ke specifikaci cesty k souboru. Standartně je výstup XmlTextWriteru nezformátovaný. Všechny elementy se zapisují za sebe do jednoho řádku, což nepůsobí vůbec pěkně a u větších XML souborů je to dosti nepřehledné. K předejití této situace má XmlTextWriter vlastnost Formatting, kterou použijeme na řádku 3 výpisu zdrojového kódu.

Funkce zálohuje v cyklu všechny tabulky z relační databáze. To znamená, že kořenový element se musí zapsat ještě před tímto cyklem. Jinak by se zapsal při zápisu každé tabulky z relační databáze a výsledný XML dokument by nebyl validní. Toto ošetření lze vidět na řádku 4. Poté se zapíšou veškeré požadované data. K zápisu elementu použijeme funkci WriteStartElement() a WriteEndElement(). Tímto ošetříme pouze zápis začínajících a koncových tagů, ale nikoliv samotných dat. K tomuto účelu slouží funkce WriteString(). Pokud je daná buňka tabulky prázdná, zapíše se pouze prázdný element.

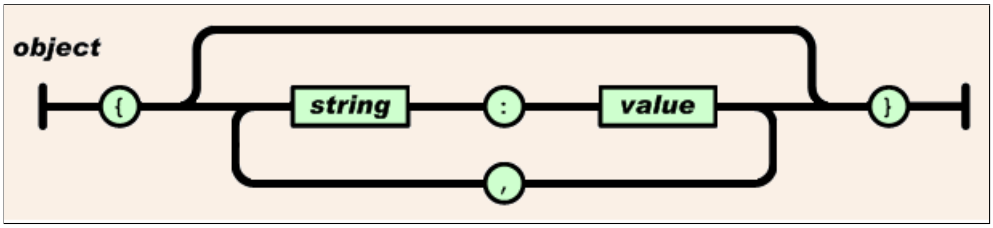
2.2 JSON

JSON byl vytvořen a popularizován americkým programátorem a zároveň podnikatelem Douglasem Crockfordem. JSON je podmnožinou JavaScriptu. Byl poprvé použit v roce 2001 ve společnosti State Software, kterou spoluzaložil právě Douglas Crockford. Často bývá nazýván odlehčenou formou a alternativou XML. Je popsán a standardizován ve Standardu ECMA-262 3rd Edition - December 1999.

JSON se částečně používá v technologii Ajax, která slouží pro komunikaci se servery. Jako příklad můžeme uvést situaci, když uživatel píše požadavek do nějakého vyhledávače. Ajax postupně našeptává možné odpovědi ze serveru, které alespoň z části odpovídají hledanému výrazu. Pro tyto aktualizace mezi serverem a klientem používají prohlížeče standardně XML, ale už se pro tento účel začal používat i JSON.

2.2.1 Základy syntaxe

JSON má podobnou strukturu jako XML. Lze použít kolekci typu název-hodnota s tím rozdílem, že JSON nemá koncové tagy jako XML. Ta v různých jazycích bývá realizována jako objekt.



Obrázek 2: Kolekce název-hodnota

2.2.2 Výhody a nevýhody

Oproti jiným formátům má JSON několikásobně menší velikost. Je jednoduše čitelný i zapisovatelný člověkem a snadno analyzovatelný i generovatelný strojově.

Co se týče nevýhod z hlediska bakalářské práce, JSON není tak rozšířen jako XML. Hlavně co se týče podpory v .NET frameworku.

2.2.3 Příklad dokumentu

Opět uvažme stejnou tabulku uživatelů z předešlého příkladu. Serializace dosáhneme třídou JsonWriter, který je součástí externí knihovny Json.NET. Možností je víc, záleží na použité knihovně. Výsledný JSON dokument bude vypadat následovně.

```
new Uzivatel(
   new Radek(
      "id_uzivatele": "buc0022",
      "jmeno": "Martin",
      "prijmeni": "Bucek",
      "email": "martinbucek@seznam.cz"
   ),
   new Radek(
      "id_uzivatele": "nov0011",
      "jmeno": "Jan",
      "prijmeni": "Novak",
      "email": "jannovak@gmail.com"
   ))
```

Výpis 3: JSON dokument

Zde můžeme vidět, že příklad tohoto dokumentu zcela neodpovídá použití kolekci název-hodnota z obrázku 2 výše. Zmínil jsem nedostatečnou podporu JSONu v .NET frameworku. Abych mohl vůbec uskutečnit serializaci do JSON formátu, musel jsem stáhnout nějakou externí DLL knihovnu. Použil jsem knihovnu JSON.NET a jako jedinou alternativu k

objektu kolekce název-hodnota jsem našel tzv. konstruktory. Proto je dokument trochu odlišný.

Následná funkce pro serializaci vypadá následovně.

```
1  using (FileStream fs = new FileStream(" ../../ Zalohy/Json.json", FileMode.Create))
2  using (StreamWriter sw = new StreamWriter(fs))
3  using (JsonWriter jw = new JsonTextWriter(sw))
4  {
5          jw.Formatting = Newtonsoft.Json.Formatting.Indented;
6          jw.WriteStartConstructor(tableName);
7          foreach (DataRow row in tableOfContents.Rows)
8          {
9                  jw.WriteStartConstructor("Radek");
10                 foreach (DataColumn column in tableOfContents.Columns)
11                 {
12                         jw.WriteRawValue(new JsonStringValue(column.ColumnName.ToString(), row[
                               column].ToString().Trim()).ToString());
13                 }
14                 jw.WriteEndConstructor();
15         }
16         jw.WriteEndConstructor();
17 }
```

Výpis 4: Serializace do JSON formátu

Funkce používá k zápisu JsonWriter, který jako parametr přijímá jakýkoliv TextWriter. Na řádcích 6 a 9 ve výpisu zdrojového kódu se právě používá zápis výše zmíněných konstruktorů. Uvnitř konstuktorů jsou páry název-hodnota, které se zapisují pomocí metody JsonStringValue() na řádku 12, která jako parametry přijímá právě název a hodnotu. Název reprezentuje název sloupce a hodnota reprezentuje obsah buňky v daném řádku tabulky. Můžeme vidět určitou podobnost použití metody JsonStringValue() ve výpisu zdrojového kódu a příkladem kolekce z obrázku 2. Nicméně při bližším pohledu se liší v syntaxi. Místo složených závorek jsou použité klasické závorky a před každým konstruktorem se nachází slovo new pro označení začátku nové části dokumentu.

2.3 YAML

YAML navrhnuli společně v roce 2001 Clark Evans, Ingy döt Net a Oren Ben-Kiki. Jedná se o textový formát, který se používá pro uložení dat databázového typu. Je do jisté míry podobný JSONu.

2.3.1 Výhody a nevýhody

YAML má několikanásobně měnší velikost než ostatní textové formáty a navíc je jednoduše čitelný i zapisovatelný člověkem a snadno analyzovatelný i generovatelný strojově. V syntaxi používá tzv. sekvence, mapy a scalary. Lze je libovolně skládat do složitějších celků jako třeba sekvence sekvencí a do každé této sekvence přidat mapu. V konečném výsledku je dokument vcelku čitelnější než ostatní formáty.

Co se týče podpory v .NET frameworku, je na tom stejně jako JSON.

2.3.2 Základy syntaxe

Syntakticky je YAML určitě blíže JSONu než XML. To z důvodu absence koncových a prázdných tagů. YAML dokument může začínat deklarací. Je to odboba prologu u XML. Ovšem zápis vypadá uplně jinak.

```
%YAML 1.2
---
```

Za deklarací můžeme vidět jakési označení pro začátek nového dokumentu ve formě tří pomlček. Konec dokumentu je pro změnu označen třemi tečkami.

Už výše jsou částečně zmíněny základní prvky syntaxe YAML. Pojďme se na ně podívat detailněji. Základním prvkem YAML dokumentu je tzv. scalar. Není to nic jiného než pouhá proměnná typu string. V dokumentu může být ohraničena úvozovkami, ale není to podmínka.

Dalším prvkem dokumentu je sekvence. Poznáme jej tak, že začíná vždy pomlčkou. Sekvence můžeme skládat dohromady s dalšími sekvencemi a vytvořit tak složitější strukturu. Příklad takových sekvencí je vysvětlen v další podkapitole. Poslední důležitý prvek syntaxe je mapa. Je to klasická konstrukce klíč-hodnota, které známe z různých programovacích jazyků jako slovník nebo asociativní pole.

```
Jméno: Martin
Příjmení: Buček
```

YAML toho nabízí co se týče syntaxe ještě více, ale pro úvod bude tohle stačit. V dalších podkapitolách nejsou jiné prvky používány.

2.3.3 Příklad dokumentu

Použijeme opět stejnou tabulku uživatelů. Jelikož YAML není standardně podporován v .NET frameworku, musíme použít opět nějakou externí DLL knihovnu.V našem případě to bude knihovna snázevm YamlSerializer. K výsledné serializaci použijeme YamlWriter. Na výstupu dostaneme tento YAML dokument.

```
1  ---
2  - Uzivatel
3  - - id_uzivatele:  buc0022
4  - - - jmeno: Martin
5  - - - prijmeni: Bucek
6  - - - email: martinbucek@seznam.cz
7  - - id_uzivatele:  nov0011
8  - - - jmeno: Jan
9  - - - prijmeni: Novak
10 - - - email: jannovak@gmail.com
11 ...
```

Výpis 5: YAML dokument

V tomto dokumentu lze vidět použití výše uvedených konstrukčních prvků, které YAML nabízí. Název tabulky Uzivatel na druhém řádku je scalar, ale také zároveň sekvence. Tedy zbytek dokumentu patří pod tuto sekvenci, což vytvoří ve výsledku vcelku přehlednou hierachii. Níže v dokumentu je použita konstrukce sekvence sekvencí a mapy. Mapy jsou velice podobné kolekci název-hodnota jako u JSONu.

Funkce pro serializaci do YAML formátu vypadá takto.

```
1  foreach (string tableName in _list)
2  {
3          command = new SqlCommand("Select * from " + tableName + ";", connection);
4          adapter = new SqlDataAdapter(command);
5          DataTable tableOfContents = new DataTable();
6          adapter.Fill(tableOfContents);
7          var yaml = new YamlSequence();
8          YamlScalar scalar = new YamlScalar(tableName);
9          yaml.Add(scalar);
10         int i = 0;
11         foreach (DataRow row in tableOfContents.Rows)
12         {
13                 i = 0;
14                 foreach (DataColumn column in tableOfContents.Columns)
15                 {
16                         if (i == 0)
17                         {
18                                 yaml.Add(new YamlSequence(new YamlMapping(column.ColumnName.
                                           ToString(), row[column].ToString().Trim())));
19                         }
20                         else
21                         yaml.Add(new YamlSequence(new YamlSequence(new YamlMapping(column.
                                   ColumnName.ToString(), row[column].ToString().Trim()))));
22                         i++;
23                 }
24         }
25         yaml.ToYamlFile("../../Zalohy/" + tableName + ".yaml");
26 }
```

Výpis 6: Serializace do YAML formátu

Tato serializace funguje odlišněji než u předešlých formátů. Veškeré požadované data ze serveru se nejprve musí nahrát do libovolné pomocné proměnné typu var a poté se pomocí funkce ToYamlFile() serializují do YAML souboru. Můžeme si všimnout na řádku 7 výpisu zdrojového kódu, že proměnná je deklarována přímo jako sekvence. Díky tomu je celý dokument uložen jako jedna velká sekvence. Poté se postupně do pomocné proměnné přidávají jednotlivé data.

Můžeme vidět také názornou ukázku výše zmíněného skládání jednotlivých konstrukčních prvků YAML. Na řádku 8 je přidán scalar jako název tabulky. Do dokumentu se zapíše jako sekvence, protože je to první prvek v dokumentu a celý dokument je jedna větší sekvence. Při bližším pohledu tato konstrukce připomíná kořenový element v XML. Dále na řádcích 16 až 18 se spojuje sekvence s mapou. Zároveň je takto řešeno oddělení jednotlivých záznamů tabulky pomocí jednoduchého čítače. A na zbývajících řádcích se skládá sekvence se sekvencí a mapou. Teoreticky by tento způsob serializace nemusel

být pro větší databáze zrovna nejrychlejší kvůli nahrávání veškerých dat do pomocné proměnné.

2.4 SQL

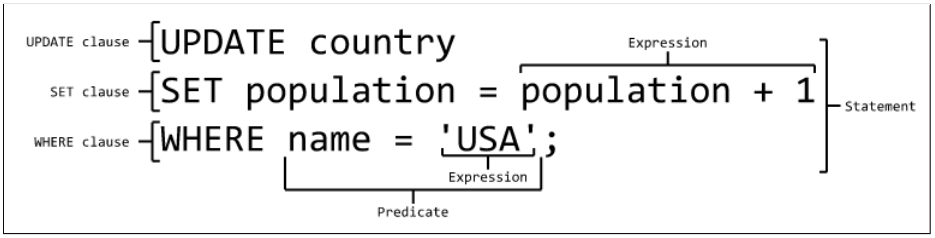
2.4.1 Původ

SQL je zcela jednoznačně nejstarší ze všech výše zmíněných formátů. Prakticky se jedná o dotazovací jazyk, ale lze jej pojmout také jako textový formát. Jeho historie se datuje až do 70.let minulého století. SQL byl vyvinut ve společnosti IBM pány Donaldem D. Chamberlinem a Raymondem F. Boycem. První verze měla název SEQUEL. Byl používán v R Systémech.

Osmdesátá léta lze považovat za počátek zlatého věku relačních databází. V roce 1987 ISO a ANSI publikují počáteční standard SQL, v roce 1989 ISO publikuje dodatek Integrity Enhancement Feature a začíná se hovořit o SQL89. V roce 1992 dochází k první upravě tohoto ISO standardu, který je pak znám jako SQL2 nebo SQL92.

První významější produkty, kde byl jazyk SQL zabudován, byly od IBM: DB2, SQL/DS (oba vycházejí ze Systému R) a QMF. S nástupem počítačů třídy PC začaly i ostatní výrobci více uvažovat o SQL. Asi nejznámější je firma Ashton-Tate se svou dBASE IV (i když implementace SQL nebyla úplná). Jako další jmenujme INFORMIX-SQL, INGRES pro PC, ORACLE, SQLBase, XDB II, XQL. Postupem času se začal jazyk SQL přenášet i na systém UNIX a od jednouživatelských úloh se přecházelo k serverům založených na SQL. Jmenujme aspoň SŘBD ORACLE, INFORMIX, PROGRESS a INGRES. Dále např. SQL Server od firmy Sybase či Microsoft.

U tohoto formátu nebude detailně popisována syntaxe jako u předešlých formátů, protože syntaxe SQL je rozšířena a známa.



Obrázek 3: Syntaxe SQL

2.4.2 Příklad dokumentu

```
1 INSERT INTO Uzivatel(login, jmeno, prijmeni, email) values('buc0022','Martin', 'Buček','
      martinbuck@seznam.cz');
2 INSERT INTO Uzivatel(login, jmeno, prijmeni, email) values('nov0011','Jan', 'Novák','
      jannovak@gmail.com');
```

Výpis 7: SQL formát

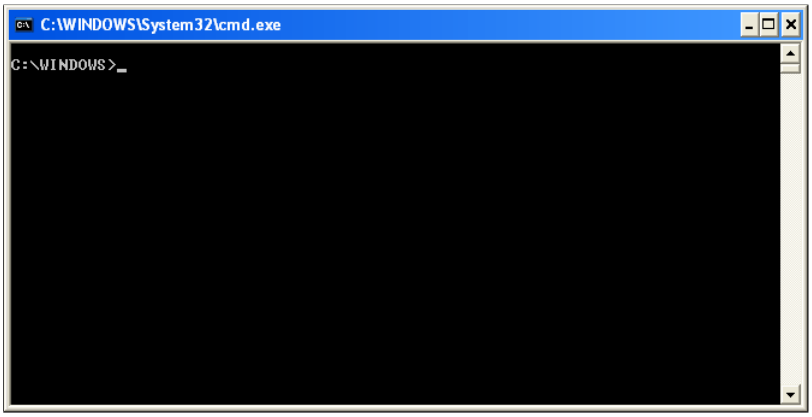
Jednotlivé záznamy jsou uloženy v podobě klasických insertů. Ovšem SQL není příliš vhodný formát pro tuto bakalářskou práci. Při několika tisíci záznamech by množství insertů bylo obrovské a proces serializace by nebyl zrovna nejefektivnější zejména kvůli velmi redundantní syntaxi SQL.

3 Ukázkový příklad použití

Cílem této kapitoly je čtenáře seznámit s ukázkovým příkladem použití vytvořeného nástroje. V této kapitole bude probírán účel nástroje a také krok po kroku návod k jeho použití. Budou také vysvětleny okrajové situace, které mohou v ukázkovém příkladu nastat. Celá tato sekce pracuje se smyšlenou databází, která je záměrně triviální kvůli přehlednosti. Po přečtení této kapitoly by čtenář měl být schopen nástroj bez problému spustit a používat.

3.1 Úvod do aplikace

Nástroj je naimplementován jako CLI aplikace v jazyce C#. Tudíž jeho spuštění je možné realizovat pouze přes příkazový řádek v operačním systému. U uživatele se předpokládá alespoň základní znalost práce v příkazovém řádku pomocí jednoduchých příkazů. Pro implementaci bylo vybráno vývojové prostředí Visual Studio 2010 od Microsoftu.



Obrázek 4: CLI rozhraní

Jelikož cílem nástroje je zálohování a obnovat dat z libovolného databázového serveru, uživatel musí mít před spuštěním potřebné údaje k úspěšnému připojení na tento server. Po konzultaci s vedoucím bakalářské práce byl vybrán Microsoft SQL Server. Uživatel se tedy bude muset omezit pouze na práci s tímto typem serveru. Nástroj bude schopen plně fungovat jedině při úspěšném připojení uživatele na server.

3.2 Práce s příkazovým řádkem

Abychom mohli aplikaci spustit přes příkazový řádek, musíme vědět kde se ona aplikace nachází. Pro jednoduchost tohoto ukázkového příkladu uvažujme, že je složka s aplikací uložena v kořenovém adresáři na disku C:\. Pro přesunutí do složky aplikace v příkazovém řádku použijeme následující příkaz:

cd *cesta*

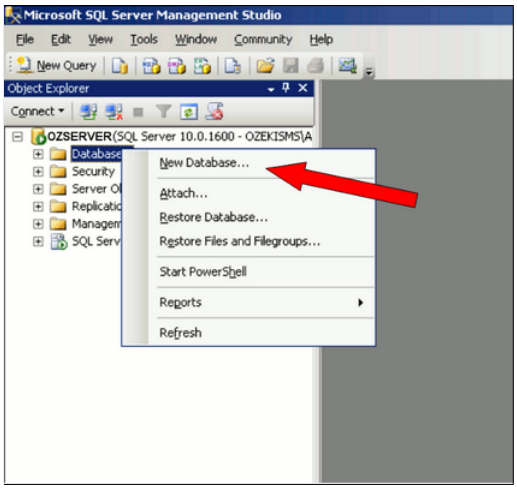
Tento příkaz přijímá jako parametr cestu k souboru. Proto je lepší si najít nejprve klasicky v operačním systému danou složku a zkopírovat cestu přímo do příkazového řádku jako parametr k příkazu.

Je možné, že před samotným spuštěním se budete v příkazovém řádku nacházet na jiném disku než je C. V tom případě vám nebude výše uvedený příkaz fungovat. Musíte se přesunout na daný disk. Pro přechod na tento disk stačí napsat následující příkaz a potvrdit klávesou Enter:

```
C:
```

3.3 Vytvoření databáze

Nástroj funguje tím způsobem, že zálohuje všechny tabulky z dané databáze. Přihlašte se tedy na svůj účet na libovolném Microsoft SQL serveru 2008 nebo vyšším a vytvořte prázdnou databázi. Nutno podotknout, že k vytvoření nové databáze musíte mít přístupová práva. V opačném případě databázi nevytvoříte. Dále stojí za zmínku se připojit do virtuální privátní sítě, pokud je to nutné.



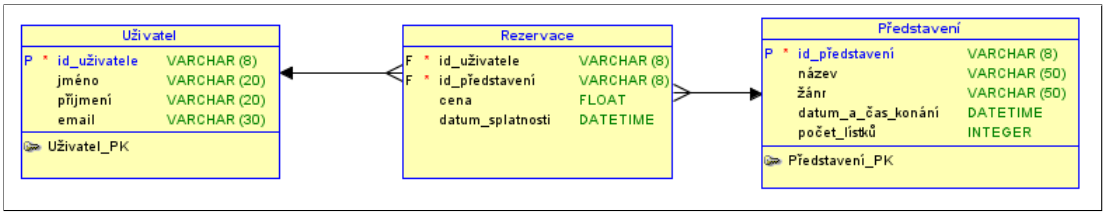
Obrázek 5: Vytvoření databáze

3.4 Nahrání inicializačního skriptu

Nyní když máme vytvořenou prázdnou databázi měli bychom ji naplnit nějakými tabulkami, aby měl nástroj s čím pracovat. V přílohách této práce najdete složku s SQL skripty.

V tomto kroce nás bude zajímat pouze skript s názvem základ.sql. Tento skript vytvoří velice jednoduchou databázi. Tato databáze má pouze tři tabulky. Reprezentuje jednoduchý rezervační systém nějakého fiktivního divadla. Navazuje částečně na tabulku uživatelů z předchozí kapitoly, která se zabývala textovými formáty. Uživatelé si mohou rezervovat místa na dané představení. Libovolný počet uživatelů si může zarezervovat

libovolný počet představení. Z toho vyplývá, že mezi tabulkami uživatele a představení bude M:N vazba a tabulka rezervace je vazební tabulka.



Obrázek 6: ER Diagram

Prohlédněte si ER Diagram dané databáze. Zejména kvůli porovnání s pozdějším výstupem aplikace. Tento skript také dané tabulky naplní. V případě problémů lze využít skript pro smazání všech tabulek z relační databáze a nahrát vše znova.

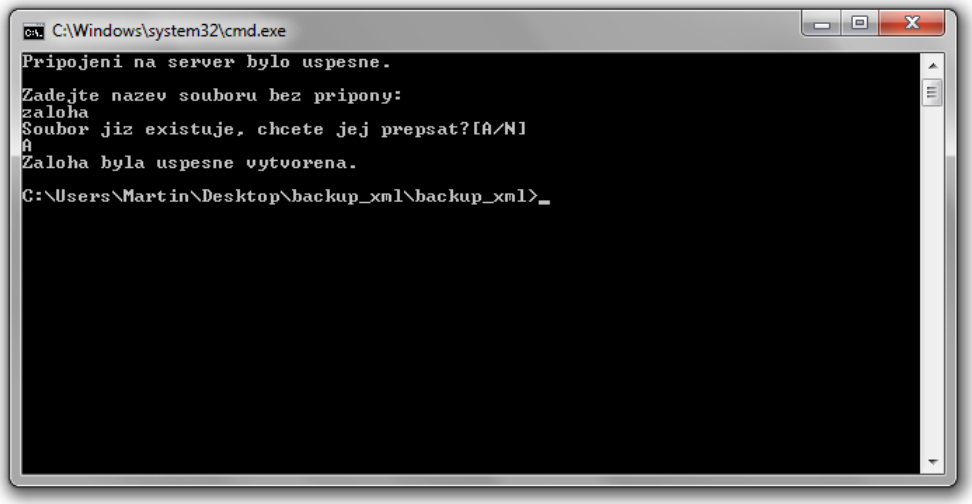
3.5 Zálohování databáze

Nyní se konečně dostáváme k využití daného nástroje. Pro zálohování použijeme dávkový soubor zaloha.bat, který se nachází ve složce s aplikací. Tento soubor přijímá 4 parametry, které jsou nutné k úspěšnému přihlášení na váš server. Příkaz pro spuštění dávkového souboru bude vypadat následovně.

zaloha *nazev_serveru nazev_databaze login heslo*

Za parametry dosadíme údaje o serveru, který chceme zálohovat. Po zadání těchto údajů se v případě úspěšného připojení objeví informativní zpráva o úspěšném připojení na vás server a dostanete se do prostředí nástroje. Poté budete vyzváni k napsání názvu záložního XML souboru, který bude obsahovat data z vašeho serveru.

Může nastat taková situace, že zadáte název již existujícího názvu XML dokumentu. V tom případě vám nástroj nabídne možnost přepsání celého souboru anebo můžete zadat název znova. Celé se to děje v cyklu, tudíž můžete zadat i vícekrát za sebou špatný název a nic se neděje. Po potvrzení dostanete informativní zprávu o úspěšném vytvoření záložního souboru a aplikace se vypne.



Obrázek 7: Zálohování

Obrázek demostruje běh aplikace během procesu zálohování.

XML dokument se ukládá do složky Zálohy ve složce s aplikací. Zkuste si jej otevřít a zkontrolovat s daty z vaší relační databáze. Zároveň v této složce můžete naleznete k danému XML dokumentu i další XML dokument se schématem celé databáze. Tyto soubory patří vždy k sobě. Poznáte jej podle sufixu Schema, který je přidán k původnímu názvu souboru. Výstupní XML dokument bude vypadat následovně.

```
<Databaze>
  <Predstaveni>
    <Radek>
      <id_predstaveni>p1</id_predstaveni>
      <nazev>Prodaná nevěsta</nazev>
      <zanr>Opera</zanr>
      <datum_a_cas_konani>22.2.2013 15:55:00</datum_a_cas_konani>
      <pocet_listku />
    </Radek>
    <Radek>
      <id_predstaveni>p2</id_predstaveni>
      <nazev>Braniboři v Čechách</nazev>
      <zanr>Opera</zanr>
      <datum_a_cas_konani>22.4.2013 17:00:00</datum_a_cas_konani>
      <pocet_listku />
    </Radek>
    <Radek>
      <id_predstaveni>p3</id_predstaveni>
      <nazev>Sluha dvou pánů</nazev>
      <zanr>Komedie</zanr>
      <datum_a_cas_konani>22.7.2013 14:15:00</datum_a_cas_konani>
      <pocet_listku />
    </Radek>
    <Radek>
      <id_predstaveni>p4</id_predstaveni>
```

```
26          <nazev>Lakomec</nazev>
27          <zanr>Komedie</zanr>
28          <datum_a_cas_konani>22.1.2013 18:00:00</datum_a_cas_konani>
29          <pocet_listku />
30        </Radek>
31      </Predstaveni>
32      <Rezervace>
33        <Radek>
34          <id_uzivatele>buc0022</id_uzivatele>
35          <id_predstaveni>p1</id_predstaveni>
36          <cena>500</cena>
37          <datum_splatnosti>21.2.2013 15:55:00</datum_splatnosti>
38        </Radek>
39        <Radek>
40          <id_uzivatele>buc0022</id_uzivatele>
41          <id_predstaveni>p2</id_predstaveni>
42          <cena>750</cena>
43          <datum_splatnosti>21.4.2013 17:00:00</datum_splatnosti>
44        </Radek>
45        <Radek>
46          <id_uzivatele>mic0109</id_uzivatele>
47          <id_predstaveni>p4</id_predstaveni>
48          <cena>500</cena>
49          <datum_splatnosti>11.6.2013 13:30:00</datum_splatnosti>
50        </Radek>
51        <Radek>
52          <id_uzivatele>nov0011</id_uzivatele>
53          <id_predstaveni>p3</id_predstaveni>
54          <cena>1000</cena>
55          <datum_splatnosti>21.7.2013 14:15:00</datum_splatnosti>
56        </Radek>
57      </Rezervace>
58      <Uzivatel>
59        <Radek>
60          <id_uzivatele>buc0022</id_uzivatele>
61          <jmeno>Martin</jmeno>
62          <prijmeni>Bucek</prijmeni>
63          <email>martinbuck@seznam.cz</email>
64        </Radek>
65        <Radek>
66          <id_uzivatele>mic0109</id_uzivatele>
67          <jmeno>Miroslav</jmeno>
68          <prijmeni>Mikus</prijmeni>
69          <email>miroslavmikus@seznam.cz</email>
70        </Radek>
71        <Radek>
72          <id_uzivatele>mik0103</id_uzivatele>
73          <jmeno>Pavel</jmeno>
74          <prijmeni>Michálek</prijmeni>
75          <email>pavelmichalek@email.cz</email>
76        </Radek>
77        <Radek>
78          <id_uzivatele>nov0011</id_uzivatele>
79          <jmeno>Jan</jmeno>
```

```
80          <prijmeni>Novak</prijmeni>
81          <email>jannovak@gmail.cz</email>
82        </Radek>
83      </Uzivatel>
84    </Databaze>
```

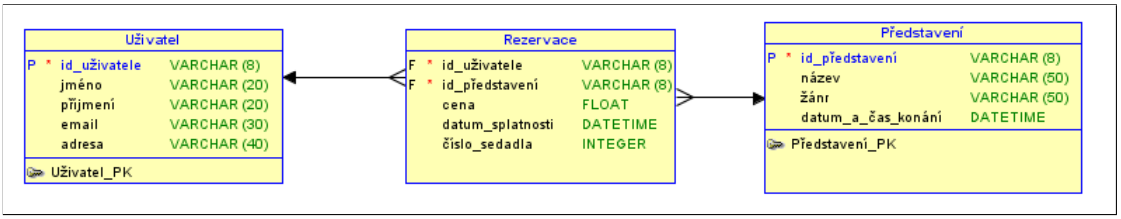
Výpis 8: Ukázkový XML dokument

XML formát je detailně popsán v první kapitole této bakalářské práce. Proto se tato sekce nebude dále zabývat rozborem tohoto XML dokumentu.

3.6 Úprava struktury databáze

V této fázi máme zálohu v XML formátu a také vytvořenou databázi na serveru. Prakticky nám nic nebrání v použití další funkce nástroje, obnovy dat. Nicméně nástroj je rozšířen také o možnost transformace databáze. Té se využívá v situaci, pokud oproti původnímu schématu databáze sloupec ubyl nebo přibyl. Pro ověření tohoto rozšíření nástroje se opět podívejte do složky se skripty. Zde naleznete skript uprava.sql.

Při bližším prozkoumání tohoto skriptu zjistíte, že maže data a upravuje samotné schéma ukázkové databáze. Některým tabulkám přidává sloupce a některým zase sloupce ubírá. Výslednou úpravu můžeme vidět v ER Diagramu. V této chvíli je důležité, aby jste nespustili znovu proces zálohování. Tím by byl myšlený příklad ztracen, protože aplikace by si uložila nové schéma a transformace by neproběhla.



Obrázek 8: ER Diagram po úpravě

V ER Diagramu můžeme vidět v každé tabulce změnu. V tabulce uživatelů přibyl sloupec adresa. Dále v tabulce rezervací přibyl sloupec číslo sedadla a v poslední tabulce ubyl sloupec počet lístků.

3.7 Obnova dat

Transformace je za náma a nyní se můžeme pustit do obnovy dat zpět na server. V příkazovém řádku se přemístěte opět do složky s aplikací a najděte dávkový soubor obnova.dat. Opět jej spusťte stejným způsobem jako v předešlém případě při spouštění dávkového souboru zaloha.dat. Příkaz tedy bude vypadat velice podobně.

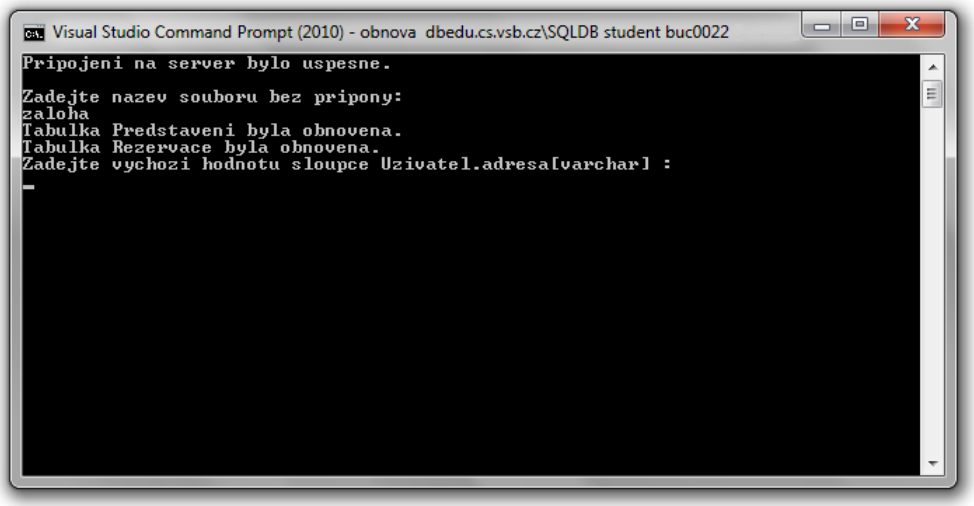
obnova *nazev_serveru nazev_databaze login heslo*

V tomto bodě může nastat několik situací. V případě zadání špatných údajů vás aplikace informuje o neúspěchu připojení na server a vypne se. Může také nastat taková situace, že se úspěšně přihlásíte na svůj server, ale z nějakého důvodu není ještě vytvořený žádný záložní XML dokument. Aplikace by neměla s čím pracovat a celý proces by skončil chybou a spadnutím celé aplikace. K ošetření tohoto případu vás aplikace informuje o této situaci a musíte nejprve vytvořit zálohu.

V případě úspěšného připojení na server a existence zálohy vás aplikace informuje o úspěchu a ihned vás vyzve k zadání názvu souboru k obnově dat. Po správném zadání názvu souboru vás systém informuje o tabulkách, které byly obnoveny. Zde se projeví výše zmíněná transformace databáze našeho ukázkového příkladu.

Je dobré se podívat také na schéma databáze k lepšímu porozumění. V představení ubyl sloupec počet lístků. S tím si dokáže nástroj poradit. Při čtení XML dokumentu bude tento sloupec ignorován ve všech záznamech této tabulky. Sloupec číslo sedadla, který přibyl v tabulce rezervace podporuje NULL hodnoty. Tento sloupec nástroj řešit nemusí. Ovšem v tabulce uživatelů přibyl sloupec adresa, který nepodporuje NULL hodnoty. U těchto sloupců musíme rozlišovat, zda má nebo nemá nastavenou výchozí hodnotu. V prvním případě se nástroj zachová stejně jako u sloupce s NULL hodnotami. Při obnově dat se automaticky doplní jeho nastavená výchozí hodnota do všech nově vložených záznamů dané tabulky. Tím se předejde chybám a nástroj tento sloupec nemusí řešit. Ovšem v druhém případě SŘBD při obnově nezná výchozí hodnotu daného sloupce. Vyřeší to tím způsobem, že při obnově vloží do všech nově vložených záznamů dané tabulky NULL hodnotu, což automaticky vyústí v chybu a obnova se neprovede.

Avšak nástroj tuto situaci dokáže ošetřit. V takovéto situaci vás před obnovou nástroj vyzve k zadání výchozí hodnoty. Informativně je v závorkách uveden i datový typ daného sloupce, který má být zadán. Poté po zadání výchozí hodnoty sloupce nástroj pozmění schéma databáze a nic už nebrání kompletní obnově dat.



Obrázek 9: Obnova dat

Obrázek demonstruje běh nástroje během obnovy. Můžeme vidět informativní zprávy o úspěšně obnovených tabulkách a hlavně výzvu k zadání výchozí hodnoty pro sloupec adresa v tabulce uživatelů. Po potvrzení se objeví zpráva o úspěšné obnově dat tabulky uživatelů a aplikace se vypne. Tímto je obnova kompletní.

Pro kontrolu funkčnosti nástroje se můžete přihlásit na váš server a zkontrolovat data v tabulkách. V tabulce uživatelů ve všech záznamech ve sloupci adresa by měla být vámi zadaná výchozí hodnota. Ve sloupci číslo sedadla v tabulce rezervací by měla být ve všech záznamech NULL hodnota.

Pro udržení konzistence zálohy a dat na serveru se po procesu obnovy spustí znova zálohování celé databáze i jejího schématu. Uživatel to běžně nepozná, protože nástroj o tom nijak neinformuje. Poté při další obnově by celý proces proběhl bez transformace plně automaticky. Můžete si opět zkontrolovat výstupní XML soubor a podívat se na změny vzniklé transformací.

Databáze byla pro tento ukázkový příklad vybrána záměrně triviální. Aplikace je schopna pracovat s mnohem větší a robustnější databází s několika tisíci záznamy v porovnání s databází v tomto textu.

Pokud by jste spustili dávkové soubory pro zálohování a obnovu znova, proběhlo by to automatizovaně bez zadávání výchozích hodnot. V tomto bodě by ukázkový příklad proběhl bez zamýšlené transformace, protože aplikace má v této fázi uložené nové schéma databáze vzniklé transformací. Pro spuštění celého procesu znova spusťte nejprve skript pro smazání tabulek smazat.sql a pak vše opakujte podle jednotlivých kroků znova.

4 Transformace databáze

Cílem této kapitoly je detailnější rozbor transformace databáze, která byla zmíněna v předchozí kapitole. Čtenář by měl po přečtení této kapitoly plně porozumět transformaci i její realizaci ve zdrojovém kódu.

4.1 Načtení schémat tabulek

Nejprve by bylo dobré porozumět celkovému řešení transformace ve zdrojovém kódu aplikace a potom se můžeme pustit do řešení jednotlivých situací ubytí a přidání sloupce. Řešení tranformace ve zdrojovém kódu stojí na funkci porovnávání struktury. Nejprve však musí být něco k porovnání.

Základním kamenem je funkce SchemaRestore, která při obnově dat načítá schéma databáze z XML dokumentu. Funkce nejprve načte z dokumentu názvy tabulek. Poté podle příšlušného názvu zálohováné tabulky zároveň tato funkce vytvoří schéma této tabulky i z SQL serveru a tyto dvě vytvořené tabulky předá jako 2 parametry funkci SchemaComparison, která tyto dvě tabulky porovnává.

COLUMN_NAME	IS_NULLABLE	COLUMN_DEFAULT	DATA_TYPE
login	NO		varchar
jmeno	YES		varchar
prijmeni	YES		varchar
email	NO	('seznam@cz')	varchar

Tabulka 2: Příklad tabulky se schématem

Tato tabulka je klíčová pro uskutečnění transformace. Pro její uložení použijeme v kódu třídu DataTable, která je součástí ADO.NET. Jako sloupce obsahuje pouze nutné atributy, které jsou potřebné k realizaci transformace. Kompletní schéma databáze je mnohem větší, ale pro náš případ bude stačit pouze tohle. V podstatě tato tabulka obsahuje metadata o tabulce v databázi.

Tabulky se schématem databáze se v aplikaci vytvářejí vždy pro jednu tabulku v jednom průběhu cyklu. Poté proběhne porovnání ve funkci SchemaComparison. Poté na základě porovnání obou tabulek proběhnou nebo neproběhnou určité operace, které si popíšem později a poté se zase vytvoří nová tabulka se schématem. K načtení této tabulky z XML dokumentu byl použit XmlTextReader. XmlTextReader je standardně podporován v .NET frameworku. Je odvozen od abstraktní třídy XmlReader. Představuje dopředný přístup k proudu dat XML. Jedná se o velice rychlý způsob čtení dat z XML dokumentu. Způsobů čtení dokumentu je více.

Můžeme například použít třídu XmlDocument a poté použít selekci v podobě zadání XPath výrazů pro výběr dat. Ovšem v tomto ohledu nemusí být tento způsob příliš efektivní, protože dokument se musí načíst do paměti a poté s ním lze pracovat. Načítání do paměti u tohoto způsobu je nutné, protože je potřeba znát stromovou strukturu XML dokumentu k použití XPath výrazů. XmlTextReader funguje opačně. Nemusí nic načítat do

paměti a to jej dělá velice rychlým. Pouze se posouvá od jednoho elementu k druhému a postupně všechny čte.

```
1  while (r.Read() && r.LocalName != tableName)
2  {
3          if (r.LocalName.Equals("Column"))
4          {
5                  DataRow row = table.NewRow();
6                  while (r.Read() && r.LocalName != "Column")
7                  {
8                          if (r.NodeType.Equals(XmlNodeType.Element))
9                          {
10                                 if (!r.IsEmptyElement)
11                                         row[r.LocalName] = r.ReadString();
12                                 else
13                                         row[r.LocalName] = "";
14                         }
15                 }
16                 table.Rows.Add(row);
17         }
18 }
```

Výpis 9: Načtení XML schématu

Můžeme vidět pouze fragment z kódu načítání schématu. Celá funkce je mnohem delší. XmlTextReader realizuje svůj dopředný přístup metodou Read. Respektive se díky této metodě posouvá od jednoho elementu k druhému. Pro vytvoření výše uvedené tabulky vytvoříme nový prázdný řádek tabulky a poté vložíme všechny hodnoty elementů, které se nachází ohraničeny elementem s názvem Column. Tento element slouží pro oddělení jednotlivých záznamů. Je to podobná struktura XML dokumentu jako u předchozích výpisů.

Ještě zbývá načíst schéma tabulky ze serveru. Samozřejmě druhá tabulka bude mít naprosto totožnou strukturu, co se sloupců týče. V počtu a obsahu řádků už se mohou lišit. Tabulku ze serveru získáme jednoduchým SQL dotazem. Použijeme systémový katalog.

```
SELECT COLUMN_NAME,IS_NULLABLE,COLUMN_DEFAULT,DATA_TYPE FROM
INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = '@tableName';
```

Za @tableName dosadíme jako parametr název tabulky, jejíž schéma chceme načíst ze serveru.

4.2 Porovnání schématů

Nyní máme vytvořené tabulky schémat a nástroj už má co porovnat. V této kapitole se podíváme blíže na funkci, která porovnává obě tabulky se schématem. Ta je jádrem celé transformace. Nejprve se porovná počet řádků v obou tabulkách. Tak se jednoduše zjistí zda sloupec ubyl nebo přibyl. V případě stejného počtu řádků se zavolá funkce pro obnovu dané tabulky a transformace neproběhne.

4.2.1 Sloupec ubyl

Může nastat situace, že sloupec oproti původnímu schématu ubyl. To znamená, že v tabulce načtené z XML dokumentu je o řádek více než v tabulce načtené ze serveru. V tom případě se vytvoří další tabulka, která bude obsahovat rozdílné řádky, které jsou v jedné tabulce a zároveň nejsou v tabulce načtené ze serveru. Následně se z této tabulky načtou pouze názvy sloupců do předem připravené globální generické kolekce. Poté je zavolána funkce pro obnovu dat. Při obnově se budou ignorovat všechny názvy sloupců, které jsou obsaženy v generické kolekci.

```
1  var diff = xmlTable.AsEnumerable().Where(r => !sqlTable.AsEnumerable().Select(x => x["
       COLUMN_NAME"]).ToList().Contains(r["COLUMN_NAME"])).ToList();
2  DataTable result = diff.CopyToDataTable();
```

Výpis 10: Vytvoření tabulky

Výpis demonstruje vytvoření tabulky s názvem result s rozdílnými řádky ze dvou původních tabulek. Jak už název vypovídá, XmlTable je tabulka načítána z XML souboru a SqlTable je tabulka načítána ze serveru. Jinými slovy tento kód vybere všechny řádky z XmlTable, které nejsou obsažené v tabulce SqlTable. Struktura tabulky bude opět stejná jako v předešlém případě u načítání schémat. Použijeme proměnnou typu var, do které nahrajeme potřebné záznamy. A poté přes funkci CopyToDataTable nahrajeme data do tabulky result.

4.2.2 Sloupec přibyl

Logicky může také nastat situace, že sloupec oproti původnímu schématu přibyl. V tomto případě opět vytvoříme velice podobným způsobem tabulku s názvem result jako v situaci s ubytím sloupce.

```
1  var diff = sqlTable.AsEnumerable().Where(r => !xmlTable.AsEnumerable().Select(x => x["
       COLUMN_NAME"]).ToList().Contains(r["COLUMN_NAME"])).ToList();
2  DataTable result = diff.CopyToDataTable();
```

Výpis 11: Vytvoření tabulky

Můžeme vidět, že kód je na první pohled prakticky totožný v předešlým. Při bližším pohledu si lze všimnout záměny XmlTable za SqlTable. Tento kód vybere řádky z SqlTable, které se neshodují s řádky v XmlTable. Takto zajístíme, že daná tabulka bude obsahovat pouze sloupce, které přibyly oproti původnímu schématu.

Ovšem v této fázi musí nástroj jednotlivé řádky tabulky result zanalyzovat detailněji. Pro tento případ nástroj používá další funkci, která řeší právě tuto problematiku. Prochází jednotlivé řádky této tabulky a u každého z nich vyhodnocuje zda podporuje či nepodporuje NULL hodnoty. Tento fakt je klíčový pro celý proces transformace.

Pokud sloupec podporuje NULL hodnoty, tedy hodnota sloupce IS_NULLABLE v řádku je YES, nic se neděje. V tom případě aplikace nemusí nic řešit, protože se nastaví implicitní hodnota sloupce jako NULL a nenastane žádná chyba. To byl pouze jeden

příklad. Nástroj může při průchodu tabulkou result narazit na sloupec, který nepodporuje NULL hodnoty. V tom případě se posuneme dál a nyní musíme rozlišovat, zda má sloupec nastavenou výchozí hodnotu.

Jestliže sloupec má nastavenou výchozí hodnotu, nástroj to opět nemusí řešit jako v předešlém případě. Při obnově se nastaví implicitní hodnota a nenastane chyba. Ovšem nás zajímá poslední možnost, když sloupec nemá nastavenou výchozí hodnotu. Normálně by se při obnově nastavila implicitní NULL hodnota a vše by vyústilo v chybu, protože sloupec samozřejmě nepodporuje NULL hodnoty. Nástroj řeší tuto situaci právě nástavením uživatelem zadané výchozí hodnoty daného sloupce. Po zadání výchozí hodnoty se ještě samo o sobě nic nestane ani nevyřeší. Tato hodnota se musí ještě nastavit v databázi. Toho dosáhneme následujícím SQL dotazem.

```
1 command=new SqlCommand("Alter table @tableName add constraint default_value_@column_name
      default '"@defaultValue'" for @column_name', connection);
2 command.ExecuteNonQuery();
```

Výpis 12: Nastavení výchozí hodnoty

Použijeme SqlCommand, který inicializujeme SQL příkazem. Pomocí alter table změníme námi požadovanou výchozí hodnotu. Přidáme nový constraint s názvem default_value a názvem sloupce. Proměnná defaultValue je výchozí hodnota, kterou zadá uživatel do konzole. Daný příkaz spustíme pomocí funkce ExecuteNonQuery. Tento proces opakujeme pro každý další sloupec, který nemá nastavenou výchozí hodnotu.

5 Závěr

Hlavním cílem bakalářské práce bylo vytvořit nástroj k zálohování a obnově dat. Nástroj dokáže pracovat s libovolným Microsoft SQL serverem a plnit bez problémů zadané cíle bakalářské práce. Vhodné textové formáty pro další implementaci jsme probrali v první kapitole Rovněž je nástroj rozšířen o možnost transformace. Výhodou nástroje je ošetření i situací, kdy ubylo nebo přibylo více sloupců najednou. S detailnějším rozborem transformace jsme byli seznámeni v ukázkovém příkladě a také v poslední kapitole. Další výhodou nástroje je plně automatizované přihlášení na server. U aplikace je také ošetřen vstup uživatele.

Nabízí se různé možnosti rozšíření nástroje. Například by mohl být v budoucnu rozšířen o možnosti rozsáhlejší manipulace se schématem libovolné databáze.

6 Reference

[1] Esposito, Dino, *XML: efektivní programování pro .NET.* Grada Publishing, 2004. ISBN 80-247-0775-6

[2] XML specifikace, http://www.w3.org/XML/

[3] XML validace, http://www.w3schools.com/xml/xml_validator.asp

[4] XmlTextWriter, http://msdn.microsoft.com/cs-cz/library/system.xml.xmltextwriter.aspx

[5] YAML, http://www.zdrojak.cz/clanky/yaml-serializacni-format-pro-ukladani-dat/

[6] YAML syntaxe, http://en.wikipedia.org/wiki/YAML

[7] JSON, http://www.json.org/json-cz.html

[8] JSON syntaxe, http://json.org/example.html

A Elektronické přílohy na CD

Zde je uvedena adresářová struktura a důležité soubory přikládaného CD.

- Bakalářská práce
  - backup_xml - složka s aplikací
    - Zálohy - složka pro XML soubory
    - Skripty - složka s SQL skripty
    - obnova.bat, zaloha.bat - dávkové soubory pro ukázkový příklad
  - BP.pdf - tištěná verze v elektronické podobě